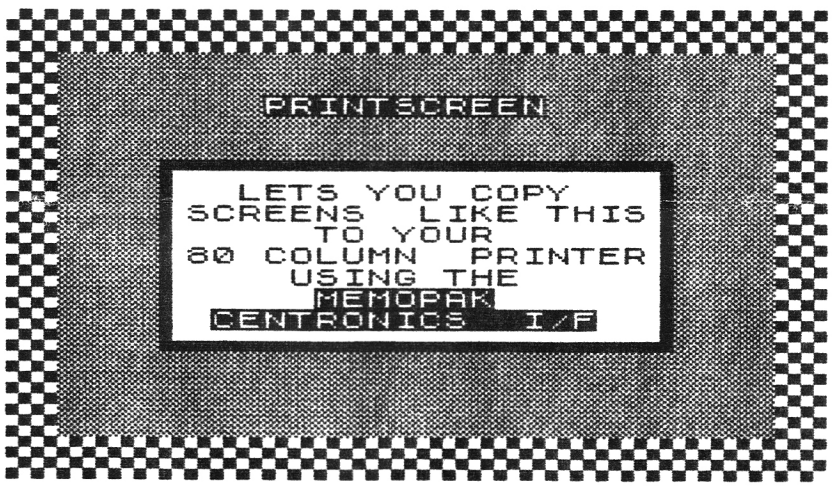
The PRINTSCREEN Utility
User Manual



PROGRAM and MANUAL Copyright
(c)1985
by Peter McMullin

All Rights Reserved, with the sole exception of rights granted to Syncware News, under the terms of Display Competition Rules.

TABLE of CONTENTS

INTRODUCTION.....1
(LOADing; what to expect)

HOW IT WORKS.....1
(elementary theory)

DIGGING DEEPER.....2
(escape codes, printer compatibility
and program mechanics)

PLAYING AROUND.....2
(modifications for fun)

APPENDICES

FIG.1 (defining Catch 22).....4

TABLE 1 (escape code data).....4

SAMPLE DEMO OUTPUT.....5

INVERT LISTING.....5

PRTSCR LISTING.....6-10

The MEMOPAK PrintScreen Utility

c1985, Peter McMullin

INTRODUCTION

PRTSCR is a 324 byte Machine-Code utility for the TS1000/ZX81 with at least 4K RAM, a MEMOPAK CENTRONICS I/F, and any standard printer with graphics capabilities.

PRTSCR lets you copy ZX characters to your Centronics printer, just as they appear on the screen!

As supplied, the program is configured to work with EPSON MX80/RX80/FX80, Mannesman Tally, Star Gemini 10X, and any other printer which uses the EPSON graphics escape codes. (This is easy to modify)

The program also contains a 19 byte INVERT routine, which can be helpful when creating printer graphics on your screen. The INVERT routine is invoked by RAND USR 16840.

TO LOAD the program, just ENTER LOAD "" or LOAD "PRTSCR"
PRTSC is autoRUN, so have your printer on-line.
The program, when loaded, will LIST itself, and print the ZX character set. (You can BREAK at any time, including during printout)
Then, it will call PRTSC, then INVERT, then PRTSC again. This will quickly show you the capabilities of the program!

You can delete all lines from line 10 on, and add your own program after viewing the demo.

How It Works

PRTSCR uses the character generator in the ROM to define bytes which it sends to the printer in graphics mode. It copies the current screen, i.e. Display File to the printer. To do this, it must find the Display File, and for 22 lines of 32 characters, it locates the dot pattern in the ROM for each character, and outputs the 8 bytes which define the character.

The routine, invoked by RAND USR 16528, first sends escape codes to the printer which set line spacing to 8 dots vertical, and set a tab so the printout will be centred on the page. Then, for each line of 32 chars., an escape code is sent to the printer which puts it in graphics mode, and specifies the number of graphic bytes per line. The line of characters is then sent, and the break key tested before proceeding to the next line. When finished, or if BREAK is pressed, the routine will reset line spacing to normal and return to BASIC with report 0/0.

This all seems straightforward enough so far, doesn't it? So why the 324 bytes? It seems we should be able to write a routine to do all this in 160 bytes or less, so what's the catch?

The heartbreaker here is this: the ROM character generator is set up in video format. Video is created by horizontal scan lines, so the characters are defined by 8 bytes each, where each byte defines a horizontal row of 8 pixel dots.

The printer, however, has a VERTICAL row of print wires, and requires that an 8-pixel by 8-pixel character must be defined by 8 bytes, each of which defines a vertical row of pixels.

Get the picture? See Fig.1 if you are confused.

So, you see, the PRTSC program not only finds the dot patterns in the ROM, but *constructs new bytes from them, bit by bit*, to create data the printer can use.

next page: MODIFYING PRTSC

DIGGING DEEPER

Printer escape codes are stored in bytes 16514 through 16526. 16527 is used as an inverse character flag, and will hold 0 or 128, depending on the last character printed. 16528 is the start of the routines.

Back now to those escape codes. See the "PRINTER ESCAPE DATA" chart for details of how these are stored.

The 7 bytes from 16514 to 16520 incl. are sent just once at the start of PRTSC. The first 3 bytes (16514-16516) set line spacing to 8 dots, while the next 4 (16517-16520) set the printer tab at the 18 col. position, which centres the printout. Study the escape data chart, and you will see how different tabs could be poked to 16519, or entirely different escape codes poked in for eccentric printers.

The next group of codes, located 16521-16526, is sent at the start of each line printed.

The first byte is the ASCII TAB flag, which causes each line to be printed at the preset tab position.

16522 thru 16526 are a graphics mode escape sequence. This sets the printer in 120DPI graphics mode for 512 bytes to follow. (There are really only 256 dots in a line, but we use high-density graphics mode and send each byte *twice* for a better looking printout.)

Normally, 5 bytes are sent from here for each line: 1-byte TAB, and the 4-byte graphics escape. If your printer uses a 5-byte graphics escape, it can be poked in starting at 16522. This would mean that 6 bytes (including the TAB) must be sent for each line. To accomplish this, POKE 16565,6 . Byte 16565 sets the number of bytes sent in the escape sequence for each line. Study of the labeled disassembly listing should make this more clear.

One escape sequence remains: when the routine finishes, it resets printer line spacing back to default setting. This is <ESC>"2" on the EPSON. The ASCII code for "2" is 50. This is located at address 16549. If the code following <ESC> for this function is other than 50 for your printer, POKE the appropriate code to address 16549.

The final printer consideration: PRTSC sends both LINEFEED and CARRIAGE RETURN codes to terminate each line. The LINEFEED code (10) is located at address 16610. The C.R. code (13) is at 16615. If you don't need one of these, POKE the appropriate address with zero.

PLAYING AROUND...

So far, the only changes discussed have dealt with modifying PRTSC to work with your wierd printer. (if you have a wierd printer)

Now for some fun-type changes. I will deal only with the EPSON codes here. If these experiments don't work, you had better check your printer manual.

In a 32 char. line, with 8-dot-wide characters, there are 256 dot columns. However, PRTSC sends these twice. Let's change this.

Bytes 16524-16525 set bytes/line in the graphics escape. 16524 is the "low byte" and holds 0. 16525 is the "high byte", and holds 2.

2*256=512: no secrets here! Enter POKE 16525,1 . This means the printer will only be expecting 256 bytes per line. That's part 1 of the change. Now, enter POKE 16815,201 .

That just put a RETurn in place of the call which outputs each byte the second time.

OK, now cross your fingers and RUN the demo.
Boy, it got narrow, didn't it! The best is yet to come!

Let's take this one step further. Thus far, we have been working in 120DPI graphics mode, <ESC>"L".

Enter POKE 16523,75. That just changed the graphics escape to <ESC>"K", which means 60DPI graphics mode. Now RUN the demo again. Wow! Back to regular width, but it looks different: not so dense, easier on the ribbon, and the inverse chars. may be more legible, depending on the state of your ribbon.

Now you have seen how easy it is to modify the size and quality of printout. If all those POKEs seem daunting, just grit your teeth and follow the instructions carefully. The absolute worst you can do is make it crash!

To summarize the POKEs we have done:

POKE 16525, 2 for 512 dots/line

POKE 16525, 1 for 256 dots/line

POKE 16815, 195 to output each byte twice(512/line)
POKE 16815, 201 to output each byte once(256/line)

POKE 16523, 76 for 120DPI graphics
POKE 16523, 75 for 60DPI graphics.

As long as the graphics escape prepares the printer for the number of bytes you send (256 or 512), you can experiment with graphics modes. Some combinations will overflow the printer line length (e.g. 512 bytes @ 60DPI).

Printers with an intermediate 72DPI graphics mode will produce an impressive printout in 512 dot/line PRTSC mode, while the 256 dot/line will give a more normal aspect ratio.

This Manual was prepared "overnight" on my ZX81, using Word Sinc II.5, WordFont II.5, PRTSC, and DCODER. (AND A PENCIL !!)

Inquiries c/o the author, at
2340 Queen St. E.
Toronto, Ont.
Canada M4E 1G9

ENJOY! PM 4/85

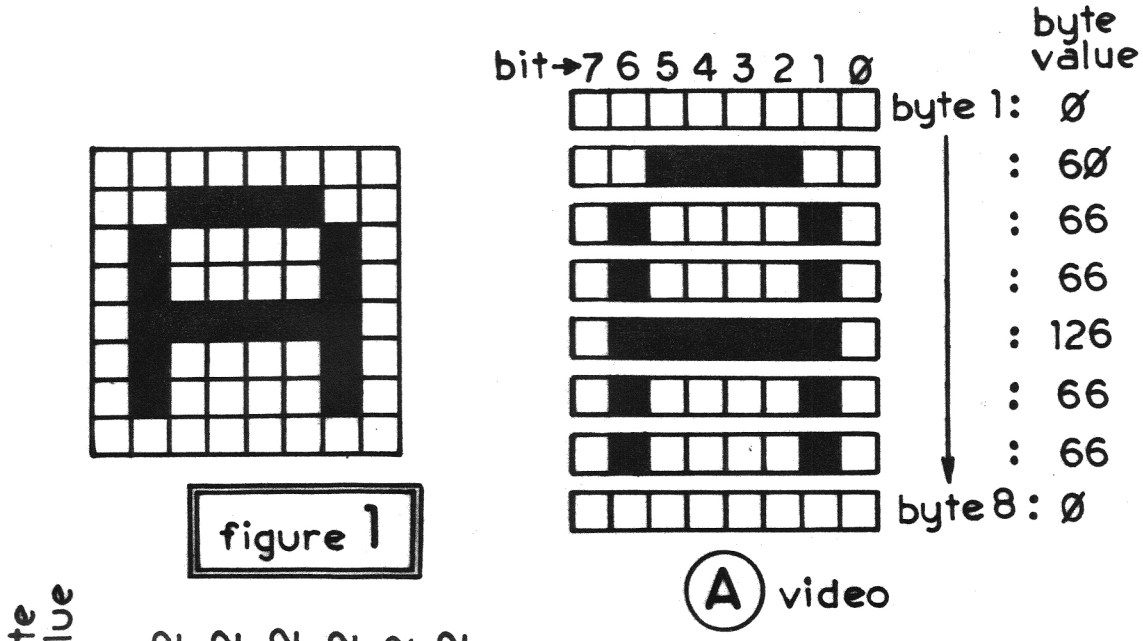


figure 1

FIG.1: Character Byte Format

1A shows how the letter "A" is defined by 8 bytes in the ROM.

1B shows how the bytes have to be re-structured so the printer doesn't print the characters sideways.

PRINTER ESCAPE DATA

16514	27	<ESC>	SET LINE SPACING
16515	65	"A"	
16516	8	8 dot line sp.	
16517	27	<ESC>	SET LINE TAB
16518	68	"D"	
16519	18	set TAB at col. 18	
16520	0	Terminate TAB seq.	
16521	9	start line @ TAB	TAB and GRAPHICS ESCAPE
16522	27	<ESC>	
16523	76	"L"	
16524	0	0+	
16525	2	2*256 bytes per line	
16526	0	spare byte	
16527	128	inverse flag	

This is the DEMO program on the tape, and this is what it does!

```
  1 REM
  0 REM      P.MCMULLIN,1985
  0 REM
  0 REM    RAND USR 16528 =
  0 REM       PRINT SCREEN
  0 REM
  7 REM   MEMOPAK I/F VERSION
  8 REM
 10 LIST 1
 20 FOR N=0 TO 191
 30 PRINT CHR$ N AND (N<64 OR N
>127);
 40 NEXT N
 50 RAND (USR 16528+USR 16840+U
SR 16528)
 60 STOP
 70 SAVE "PRTS"
 80 RUN
 "£$:?()><=+-*/;,.0123
456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
```


```
 "£$:?()><=+-*/;,.0123
456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
  1 REM
  0 REM      P.MCMULLIN,1985
  0 REM
  0 REM    RAND USR 16528 =
  0 REM       PRINT SCREEN
  0 REM
  7 REM   MEMOPAK I/F VERSION
  8 REM
 10 LIST 1
 20 FOR N=0 TO 191
 30 PRINT CHR$ N AND (N<64 OR N
>127);
 40 NEXT N
 50 RAND (USR 16528+USR 16840+U
SR 16528)
 60 STOP
 70 SAVE "PRTS"
 80 RUN
 "£$:?()><=+-*/;,.0123
456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
 "£$:?()><=+-*/;,.0123
456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
```


The INVERT Routine (RAND USR 16840)

```
          addr  byte  mnemonic     (DIS)
          ------------------------------
START --> 16840 |2A| LD   HL, (16396) -- (DFILE)
          16841 |0C|
          16842 |40|
          16843 |0E| LD   C , 22 -- LINES
          16844 |16|
LINE  --- 16845 |06| LD   B , 32 -- CHARS/LINE
          16846 |20|
CHAR. --- 16847 |23| INC  HL     -- SKIP N/L , INC CHAR.
          16848 |7E| LD   A , (HL)  } GET CODE
          16849 |C6| ADD  A , 128   } ADD 128
          16850 |80|                } PUT IT BACK
          16851 |77| LD   (HL), A   }
          16852 |10| DJNZ -7       NEXT CHAR
          16853 |F9|
          16854 |23| INC  HL     - SKIP N/L CODE
          16855 |0D| DEC  C
          16856 |20| JR   NZ, -13 NEXT LINE
          16857 |F3|
          16858 |C9| RET  --- BACK TO BASIC
```

```
        addr  byte  mnemonic      (DIS)
        ttttttttttttttttttttttttttttttt
START   16528 |21| LD   HL, 16514
        16529 |82|
        16530 |40|
        16531 |06| LD   B , 7
        16532 |07|
        16533 |7E| LD   A , (HL)
        16534 |CD| CALL 10549
        16535 |35|
        16536 |29|
        16537 |23| INC  HL
        16538 |10| DJNZ -7
        16539 |F9|
        16540 |CD| CALL 16553
        16541 |A9|
        16542 |40|
FINISH- 16543 |3E| LD   A , 27
        16544 |1B|
        16545 |CD| CALL 10549
        16546 |35|
        16547 |29|
        16548 |3E| LD   A , 50
        16549 |32|
        16550 |C3| JP   10549
        16551 |35|
        16552 |29|
MAIN-   16553 |2A| LD   HL, (16396)
        16554 |0C|
        16555 |40|
        16556 |06| LD   B , 22
        16557 |16|
LINE-   16558 |C5| PUSH BC
        16559 |23| INC  HL
        16560 |E5| PUSH HL
        16561 |21| LD   HL, 16521
        16562 |89|
        16563 |40|
        16564 |06| LD   B , 5
        16565 |05|
        16566 |7E| LD   A , (HL)
        16567 |CD| CALL 10549
        16568 |35|
        16569 |29|
        16570 |23| INC  HL
        16571 |10| DJNZ -7
        16572 |F9|
        16573 |E1| POP  HL
        16574 |06| LD   B , 32
        16575 |20|
CHAR.   16576 |C5| PUSH BC
        16577 |7E| LD   A , (HL)
        16578 |CB| BIT  7 , A
        16579 |7F|
        16580 |20| JR   NZ, 11
        16581 |0B|
NORMAL  16582 |E5| PUSH HL
        16583 |21| LD   HL, 16527
        16584 |8F|
        16585 |40|
        16586 |CB| RES  7 , (HL)
        16587 |BE|
        16588 |CD| CALL 16628
        16589 |F4|
        16590 |40|
        16591 |18| JR   11
        16592 |0B|
INVERSE 16593 |CB| RES  7 , A
        16594 |BF|
        16595 |E5| PUSH HL
        16596 |21| LD   HL, 16527
        16597 |8F|
        16598 |40|
        16599 |CB| SET  7 , (HL)
        16600 |FE|
        16601 |CD| CALL 16628
        16602 |F4|
        16603 |40|
        16604 |E1| POP  HL
        16605 |23| INC  HL
        16606 |C1| POP  BC
        16607 |10| DJNZ -33
        16608 |DF|
        16609 |3E| LD   A , 10
        16610 |0A|
        16611 |CD| CALL 10549
        16612 |35|
        16613 |29|
        16614 |3E| LD   A , 13
        16615 |0D|
        16616 |CD| CALL 10549
        16617 |35|
        16618 |29|
        16619 |C1| POP  BC
        16620 |CD| CALL 3910
        16621 |46|
        16622 |0F|
        16623 |30| JR   NC, -82
        16624 |AE|
        16625 |10| DJNZ -69
        16626 |BB|
        16627 |C9| RET
FIND-   16628 |21| LD   HL, 7672
        16629 |F8|
        16630 |1D|
        16631 |3C| INC  A
        16632 |47| LD   B , A
        16633 |11| LD   DE, 8
        16634 |08|
        16635 |00|
        16636 |19| ADD  HL, DE
        16637 |10| DJNZ -3
        16638 |FD|
GET     16639 |CD| CALL 16822
BITS    16640 |B6|
        16641 |41|
        16642 |CB| BIT  7 , (HL)
        16643 |7E|
        16644 |EB| EX   DE, HL
        16645 |C4| CALL NZ, 2762
```

SEND 7 INITIAL <ESC> CODES AT 16514

CALL MAIN ROUTINE

RESTORE LINE SPACING TO NORM

- RET TO BASIC FROM THIS SUB.

MAIN - DFILE

LINES

- SAVE LINE

- SKIP N/L

- SAVE ADDR.

SEND 5 <ESC> CODES FOR LINE

- RESTORE ADDR.

- CHARS PER LINE

CHAR. - SAVE CHARS

- CHAR CODE

- INVERSE?

- INVERSE!

NORMAL - SAVE ADDR

- INVERSE FLAG

- FLAG NORMAL

- FIND CHAR. IN CHAR. GEN, OUTPUT IT

- TO NEXT CHAR

INVERSE - CHANGE TO NORM. CODE

- SAVE ADDR.

- flag

set inverse flag

- find/output

- RESTORE ADDR

- NEXT CHAR ADDR

} NEXT CHAR

LINEFEED

- I/F OUTPUT SUB

CARR. RET.

- LINE #

- BREAK PRESSED?

THEN FINISH

- NEXT LINE?

- TO FINISH

FIND - FIND START BYTE FOR CODE IN CHAR. GEN.

- SAVE HL, CLEAR A

SET BIT ADDR IN DE

GET BITS - IS BIT SET?

- CALL JP(HL)

```
16646 |CA|
16647 |0A|
16648 |CD| CALL 16832 - REG, SETUP
16649 |C0|
16650 |41|
16651 |10| DJNZ -11 - SAME BIT, NEXT BYTE
16652 |F5|
16653 |CD| CALL 16801 - OUTPUT BYTE
16654 |A1|
16655 |41|
16656 |CD| CALL 16818 - SETUP FOR NEXT BIT-CHECK SEQ
16657 |B2|
16658 |41|
16659 |CB| BIT  6 , (HL)
16660 |76|
16661 |EB| EX   DE, HL
16662 |C4| CALL NZ, 2762 => JP(HL)
16663 |CA|
16664 |0A|
16665 |CD| CALL 16832
16666 |C0|
16667 |41|
16668 |10| DJNZ -11 - next byte
16669 |F5|
16670 |CD| CALL 16801 - OUTPUT BYTE
16671 |A1|
16672 |41|
16673 |CD| CALL 16818 - SETUP
16674 |B2|
16675 |41|
16676 |CB| BIT  5 , (HL)
16677 |6E|
16678 |EB| EX   DE, HL
16679 |C4| CALL NZ, 2762
16680 |CA|
16681 |0A|
16682 |CD| CALL 16832
16683 |C0|
16684 |41|
16685 |10| DJNZ -11
16686 |F5|
16687 |CD| CALL 16801
16688 |A1|
16689 |41|
16690 |CD| CALL 16818
16691 |B2|
16692 |41|
16693 |CB| BIT  4 , (HL)
16694 |66|
16695 |EB| EX   DE, HL
16696 |C4| CALL NZ, 2762
16697 |CA|
16698 |0A|
16699 |CD| CALL 16832
16700 |C0|
16701 |41|
16702 |10| DJNZ -11
16703 |F5|
16704 |CD| CALL 16801
16705 |A1|
```

CONTINUE TO GET ALL BITS FOR CHAR, BUILD UP BYTES IN A, SEND TO PRINTER

```
16706 |41|
16707 |CD| CALL 16818
16708 |B2|
16709 |41|
16710 |CB| BIT  3 , (HL)
16711 |5E|
16712 |EB| EX   DE, HL
16713 |C4| CALL NZ, 2762
16714 |CA|
16715 |0A|
16716 |CD| CALL 16832
16717 |C0|
16718 |41|
16719 |10| DJNZ -11
16720 |F5|
16721 |CD| CALL 16801
16722 |A1|
16723 |41|
16724 |CD| CALL 16818
16725 |B2|
16726 |41|
16727 |CB| BIT  2 , (HL)
16728 |56|
16729 |EB| EX   DE, HL
16730 |C4| CALL NZ, 2762
16731 |CA|
16732 |0A|
16733 |CD| CALL 16832
16734 |C0|
16735 |41|
16736 |10| DJNZ -11
16737 |F5|
16738 |CD| CALL 16801
16739 |A1|
16740 |41|
16741 |CD| CALL 16818
16742 |B2|
16743 |41|
16744 |CB| BIT  1 , (HL)
16745 |4E|
16746 |EB| EX   DE, HL
16747 |C4| CALL NZ, 2762
16748 |CA|
16749 |0A|
16750 |CD| CALL 16832
16751 |C0|
16752 |41|
16753 |10| DJNZ -11
16754 |F5|
16755 |CD| CALL 16801
16756 |A1|
16757 |41|
16758 |CD| CALL 16818
16759 |B2|
16760 |41|
16761 |CB| BIT  0 , (HL)
16762 |46|
16763 |EB| EX   DE, HL
16764 |C4| CALL NZ, 2762
16765 |CA|
```

```
16766 |0A|
16767 |CD| CALL 16832
16768 |C0|
16769 |41|
16770 |10| DJNZ -11
16771 |F5|
16772 |CD| CALL 16801 - OUTPUT BYTE
16773 |A1|
16774 |41|
16775 |E1| POP  HL - CLEAN UP STACK
16776 |C9| RET  - TO NEXT CHAR, 16604
SET 16777 |CB| SET  7 , A
16778 |FF|
16779 |C9| RET
16780 |CB| SET  6 , A
16781 |F7|
16782 |C9| RET
16783 |CB| SET  5 , A
16784 |EF|
16785 |C9| RET
16786 |CB| SET  4 , A
16787 |E7|
16788 |C9| RET
16789 |CB| SET  3 , A
16790 |DF|
16791 |C9| RET
16792 |CB| SET  2 , A
16793 |D7|
16794 |C9| RET
16795 |CB| SET  1 , A
16796 |CF|
16797 |C9| RET
16798 |CB| SET  0 , A
16799 |C7|
16800 |C9| RET
OUT 16801 |21| LD   HL, 16527 - INVERSE FLAG
BYTE 16802 |8F|
16803 |40|
16804 |CB| BIT  7 , (HL) INVERSE?
16805 |7E|
16806 |28| JR   Z , 2 - NORMAL
16807 |02|
16808 |EE| XOR  255 - INVERT BYTE IMAGE
16809 |FF|
16810 |F5| PUSH AF
16811 |CD| CALL 10549
16812 |35|
16813 |29|
16814 |F1| POP  AF
16815 |C3| JP   10549
16816 |35|
16817 |29|
SET 16818 |D1| POP  DE - SAVE RET. ADDR
UP: 16819 |E1| POP  HL - RESTORE HL
NEXT 16820 |18| JR   1
BIT 16821 |01|
16822 |D1| POP  DE - SAVE HL,
16823 |E5| PUSH HL   RETURN ADDRESS
16824 |D5| PUSH DE
16825 |AF| XOR  A    CLEAR A
```

BIT SET SUBS

OUTPUT BYTE TWICE

```
16826 |11| LD   DE, 16777 - first SET SUB
16827 |89|
16828 |41|
16829 |06| LD   B , 8 - Bytes
16830 |08|
16831 |C9| RET
REG 16832 |EB| EX   DE, HL - SET UP
16833 |13| INC  DE        REGISTERS
16834 |13| INC  DE        TO CHECK
16835 |13| INC  DE        NEXT BIT
16836 |23| INC  HL
16837 |C9| RET

PROCESSING COMPLETED
```

ZXAS SOURCE CODE FOR PRTSC.

```
  23 REM (
DATA
   3 REM :L0DEC DE;LD B.C;EX AF.AF;DEC DE;LD B.H;LD (DE).A;NOP
   5 REM :L1ADD HL.BC;DEC DE;LD C.H;NOP;LD (BC).A;NOP
   6 REM :L2NOP
  10 REM LD HL.L0;LD B.7;:L18LD A.(HL);CALL 10549;INC HL;DJNZ.L18;CALL L10
START
  30 REM :L55LD A.27;CALL 10549;LD A.50;JP 10549
  40 REM :L10LD HL.(16396);LD B.22
  50 REM :L11PUSH BC;INC HL;PUSH HL; LD HL.L1;LD B.5;:L19LD A.(HL);CALL 11829;IN
C HL;DJNZ.L19;POP HL;LD B.32
  60 REM :L12PUSH BC;LD A.(HL);BIT 7.A;JR NZ.L15;PUSH HL;LD HL.L2;RES 7.(HL);CAL
L L20;JR L21
  90 REM :L15RES 7.A;PUSH HL;LD HL.L2;SET 7.(HL);CALL L20
 110 REM :L21POP HL;INC HL;POP BC;DJNZ.L12;LD A.10;CALL 10549;LD A.13;CALL 10549
;POP BC;CALL $F46;JR NC.L55;DJNZ.L11;RET
 115 REM :L20LD HL.7672;INC A;LD B.A;LD DE.8;:L16ADD HL.DE;DJNZ.L16

 118 REM CALL L99;:L118BIT 7.(HL);EX DE.HL;CALL NZ.$ACA;CALL L244;DJNZ.L118;CALL
 L60
 120 REM CALL L98;:L120BIT 6.(HL);EX DE.HL;CALL NZ.$ACA;CALL L244;DJNZ.L120;CALL
 L60
 130 REM CALL L98;:L130BIT 5.(HL);EX DE.HL;CALL NZ.$ACA;CALL L244;DJNZ.L130;CALL
 L60
 140 REM CALL L98;:L140BIT 4.(HL);EX DE.HL;CALL NZ.$ACA;CALL L244;DJNZ.L140;CALL
 L60
 150 REM CALL L98;:L150BIT 3.(HL);EX DE.HL;CALL NZ.$ACA;CALL L244;DJNZ.L150;CALL
 L60
 160 REM CALL L98;:L160BIT 2.(HL);EX DE.HL;CALL NZ.$ACA;CALL L244;DJNZ.L160;CALL
 L60
 170 REM CALL L98;:L170BIT 1.(HL);EX DE.HL;CALL NZ.$ACA;CALL L244;DJNZ.L170;CALL
 L60
 180 REM CALL L98;:L180BIT 0.(HL);EX DE.HL;CALL NZ.$ACA;CALL L244;DJNZ.L180;CALL
 L60;POP HL;RET
 190 REM :L31SET 7.A;RET;:L32SET 6.A;RET;:L33SET 5.A;RET;:L34SET 4.A;RET;:L35SET
 3.A;RET;:L36SET 2.A;RET;:L37SET 1.A;RET;:L38SET 0.A;RET
 195 REM :L60LD HL.L2;BIT 7.(HL);JR Z.L81;XOR 255
 200 REM :L81PUSH AF;CALL 10549;POP AF;JP 10549
 210 REM :L98POP DE;POP HL;JR L100
 220 REM :L99POP DE;:L100PUSH HL;PUSH DE;XOR A;LD DE.L31;LD B.8;RET

 230 REM :L244EX DE.HL;INC DE;INC DE;INC DE;INC HL;RET
5000 REM )
```